

HEWLETT-PACKARD COMPANY
Intellectual Property Administration
P. O. Box 272400
Fort Collins, Colorado 80527-2400

Patent Application
Attorney Docket No. 10008402-1

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

This is a U.S. Patent Application for:

Title Line #1: * E-SERVICE MANAGEMENT THROUGH
Title Line #2: * DISTRIBUTED CORRELATION

Inventor #1: * Akhil Sahai
Address: * 2861 Stevenson Street, Santa Clara, California 95051
Citizenship: * India

Inventor #2: * Jinsong Ouyang
Address: * 501 Twinwood Loop, Roseville, California 95678
Citizenship: * P.R.C.

Inventor #3: * Vijay Machiraju
Address: * 707 Continental Circle, #1619, Mountain View, California 94040
Citizenship: * India

"Express Mail" mailing label number: ET418391081US

Date of Deposit: June 19, 2001

I hereby certify that this paper or fee is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to the Commissioner for Patents, Washington, D.C. 20231.

By Judith E. Brown
Typed name: Judith E. Brown

E-SERVICE MANAGEMENT THROUGH DISTRIBUTED CORRELATION

5 TECHNICAL FIELD

The invention relates generally to methods and systems for monitoring electronic services (e-services) and more particularly to enabling distributed correlation for end-to-end determinations of e-service
10 conversations.

BACKGROUND ART

With the widespread deployment of the global communications
15 network referred to as the Internet, the utilization of e-services has become prevalent. E-services are created in the form of portals and e-business websites. E-service providers interact among themselves to extend a range of functionalities to clients. The interaction may take the form of a static composition in which trading partners are generally fixed, but the trend is
20 toward dynamic composition in which e-service providers dynamically choose their trading partners. As one example, an e-service provider of travel services may interact with a number of alternative e-service providers of airline reservations.

An "e-service" is an on-line service that completes tasks, solves
25 problems, or conducts transactions. E-services are accessible on the Internet at a particular Uniform Resource Locator (URL). An e-service provider may depend upon other e-service providers, with the cooperation being either static or dynamic in nature, as previously noted. In Fig. 1, a parent e-service provider 10 may undertake separate conversations 12 and 14 with two other
30 e-service providers 16 and 18. Provider 10 may be an on-line travel service, while the providers 16 and 18 may be airline and hotel services. The provider 16 is shown as engaging in separate conversations 20 and 22 with a pair of sub e-service providers 24 and 26. The sub providers may be competitors in a car service business. Thus, the e-service provider 16 may subcontract
35 related services.

The e-service providers are federated in nature, since they interact across management domains and enterprise networks. The implementations of the e-services may be vastly different and may be based

upon any of a number of different platforms. The diversity in their implementations is one reason why it is difficult to manage the overall process.

5 In order for the e-service providers 10, 16, 18, 24 and 26 to undertake the conversations 12, 14, 20 and 22, there must be an agreed upon document-exchange protocol or protocols. Standardized protocols include CBL, cXML, and EDI. Each conversation consists of multiple exchanges of documents, such as Extensible Markup Language (XML) documents. Each transmission of an XML document is referred to as an "exchange," so that a conversation will consist of multiple XML document
10 exchanges. In its simplest form, a conversation consists of an attempt to enter one transaction, but a conversation can result in multiple transactions or may result in no transaction.

Management of e-service conversations is a challenging task because of the varied, federated, decentralized and distributed nature of
15 e-services. As a single transaction spans multiple e-service providers, it is difficult to implement end-to-end e-service management. Nevertheless, there are motivations for enabling the end-to-end management. Such motivations arise from the perspective of both clients and e-service providers. Clients are interested in tracking their interactions and in understanding the internal
20 e-service process flow. An end-to-end understanding enables the client to seek some insight into the actual e-service flow. Service providers that are using other services to provide composite services, such as the providers 10 and 16 in Fig. 1, would benefit from recognizing how the component service providers are behaving. By understanding and observing the behaviors of
25 sub e-services, a composite e-service would be able to optimize itself by either changing its component sub providers or by instructing the existing component sub providers to improve performance.

Referring now to Fig. 2, a typical e-service receives an http request 28 from the parent e-service 10 (if it is not itself the root service) or
30 from an ultimate consumer at a client device, such as a personal computer. The request is routed from one of the web servers 30 in a web server farm 32 to one of the application modules 34 in an application server farm 36. Thereafter, a module 38 of the e-service business logic 40 is applied to the request and a new request 42 is generated for transmission to at least one sub
35 e-service 24. The request 42 is an http request. Some action is performed at the e-service 24 as a response to the request 42. The e-service 24 then generates a response 44 that is expected by a "listener" 46 attached to the web server farm 32.

After the http response 44 is received from the sub e-service 24, the response is directed to the business logic 40, which generates a second http response 48 that is transmitted to the parent e-service 10. The business logic 40 represents the day-to-day operations that are based upon the resources, capabilities and business rules of the particular e-service 16.

The communication pattern among the three e-services 10, 16 and 24 can vary depending upon the implementation. For example, the application logic at the application server farm 36 can cause an immediate response to be sent to the parent e-service 10, before receiving the response 44 from the sub e-service 24. A final response 48 may then be sent at a later time. Also, the final response can be sent directly to the parent e-service 10 from the sub e-service 24, instead of being routed through the business logic 40 of the intermediate e-service 16. In addition, the format of communication is not "symmetric" in the e-services world. That is, an e-service request may not always have a matching response or a single e-service request may have multiple responses. All of these factors add to the difficulty in monitoring and managing end-to-end e-service transactions. Since there is no single point of control, it is difficult to monitor the requests 28 and 42 and the responses 44 and 48 going into and coming from an e-service and to correlate the documents without knowing the business logic of the e-service. Therefore, e-service conversation and transaction level correlation typically requires intrusive instrumentation.

U.S. Patent No. 6,108,700 to Maccabee et al. describes a method and program storage device for correlating and collating selected measurement events into transactions that describe the behavior of end-to-end business transactions. The end-to-end business transactions are represented by all of the processing stages, such as the requests and responses, that comprise the business transaction. The Maccabee et al. method measures these processing stages and the communications between them by using sensors. The sensors monitor for selected changes in state using a variety of methods to glean which activities are being achieved. Examples of sensors include (1) software written to interact with software exits by registering for notification of selected conditions, (2) software and/or hardware written to intercept activities taken by the business transaction's software and/or hardware (e.g., interception DLLs or shared libraries, or analysis of output logs or alert messages), and (3) insertion of software and/or hardware probes within the business transaction's software and/or hardware (e.g., Application Response Measurement Application Programming

Interface (ARM API) calls within business transaction source code). When appropriate, a sensor generates an event that describes the change in state, when and where it has occurred, and any extra data necessary to uniquely identify the event. The events include any additional correlation data useful for later associating the event with other events to form transactions. The sensors forward the events that they generate to their agents for temporary storage, and in certain cases distribution to other system components that have registered interest in knowing that a particular event has occurred.

While the Maccabee et al. method provides significant insights to the end-to-end business transaction processing, what is needed is a more distributed correlation approach that enables end-to-end correlation of e-service transactions in a decentralized manner.

SUMMARY OF THE INVENTION

Managing business-to-business cooperation among multiple e-service providers is enabled by executing a number of steps at each of the e-service providers. That is, for each provider, local actions are implemented in order to allow localized correlations of provider interactions. For each e-service provider that initiates a transaction (e.g., the parent e-service), a transaction instance is registered. The registration may take place in a management information library that maintains a list of registered transactions. The registration includes assigning a unique transaction identification to the transaction. Also at each e-service provider, a tag is appended to interactions such as requests and responses. The appended tag is typically header information for a document (e.g., an XML document) that represents the interaction. The document includes the unique transaction identification that was assigned to the transaction with which the interaction is associated.

When a response is received from a remote e-service provider, the appended tag is accessed. The appended tag is configured such that its contents are accessible independently of the contents of the response. The tag includes correlation information and management information. The correlation information ("correlator") for a particular transaction instance is made unique by the combination of the assigned transaction identification, a transaction handle that is a locally incremented integer, and an interaction handle that is an incremented integer corresponding to an interaction with a sub e-service. The interaction handles of a specific transaction instance are used to identify the sub e-services' contributions to the transaction's statistics.

On the other hand, the management information is indicative of operational parameters that are specific to the responding e-service provider. The operational parameters may include an e-service health index (e.g., up, down, congested, etc.), an indication of e-service availability, an indication of reliability (e.g., the number of faults per number of handled service requests),
5 an indication of performance (e.g., response time or response status), and/or an indication of a fault which occurred when the request was serviced.

Each e-service provider which participates in a transaction updates the management information to include local operational parameters.
10 Consequently, the tag that includes the updated management information identifies a provider tree of transaction interaction and contributions. When the tags are utilized to correlate interactions, a local management information library can provide the information that is necessary for end-to-end conversation management. Moreover, a parent e-service is enabled to
15 perform sub e-service ranking and selection on the basis of factors such as reliability and performance.

As an e-service conversation is undertaken across multiple e-services, the correlation information is employed to track the conversation and to correlate the management information collected at each participating
20 e-service. The tags may be referred to as "management information structures" that include the correlator and a management information object having a policy component and an operational parameter component. Within the conversation across multiple e-services, the management information structures are correlated and updated in order to be sent along with response
25 documents. Management information at every level is collated into a single management information structure which is finally sent in a response to a parent e-service. That is, when an e-service sends its parent a response at the middle or the end of a conversation, the piggybacked management information structure contains the sub management information relevant to
30 the sub e-services involved in the conversation.

Optionally, e-services at each level can further enrich the data collection by agreeing on a certain "e-service specific management information object" (ESSMIO). Such a ESSMIO is defined in a particular
35 schema that enables the collaborating e-services to furnish business logic data, such as the number of book-buy requests that are received. As a result, both the management information and the ESSMIO are updated by an intermediate e-service prior to appending a tag to a response to be transmitted to a parent e-service.

While not critical, Management Information Application Programming Interfaces (MI APIs) may be used in registering, retrieving, and setting the tags.

5 BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a schematic view of the cooperation among e-services in order to provide a composite service, in accordance with known techniques.

10 Fig. 2 is a schematic view of components within a particular e-service that is cooperating with other e-services in accordance with known techniques.

Fig. 3 is a block diagram of components of an e-service system that is configured to implement the present invention.

15 Fig. 4 illustrates one embodiment of a management information structure that is appended to incoming and outgoing documents of the system of Fig. 3.

Fig. 5 illustrates an enhanced embodiment of a management information structure.

20 Fig. 6 is a process flow of steps that are followed by the system of Fig. 3 when a request document is received.

Fig. 7 is a process flow of steps that are followed by the system of Fig. 3 in sending a request document.

25 Fig. 8 is a process flow of steps that are followed by the system of Fig. 3 when a response document is received.

Fig. 9 is a simplified example of the operations of Figs. 6–8.

Figs. 10–14 illustrate a more complex example of the operations of Figs. 6–8.

30 DETAILED DESCRIPTION

With reference to Fig. 3, an e-service system 50 includes a web server farm 52, an application server farm 54 and business logic 56 that are conventional components in accordance with the description of Fig. 2. Thus, requests and responses that are received by a server of the web server farm 52 via a network interface 58 may be directed to a module of the application server 54, which cooperates with the proprietary business logic 56 of the system. In one embodiment, the network interface 58 provides access to the

35

global communications network referred to as the Internet, but other applications are contemplated.

The system 50 also includes a document engine 60 that is connected to the application server farm 54. The document engine generates the documents which embody interactions with other e-services 62 and 64 and with clients 66, such as personal computers of end users. While not critical, the document engine may generate XML documents.

In addition to the conventional components, the e-service system 50 includes a management information library 68 and a header generator 70. The header generator 70 forms tags that are appended to documents to be transmitted to another e-service 62 and 64 or the client 66. A problem with providing localized correlation (i.e., distributed correlation) is the fact that request and response documents are usually different documents. The management information (MI) library 68 locally maintains correlation information and updates the correlation information in the header of the documents that are provided to the local system 50. Thus, in order for the distributed correlation approach to operate effectively, cooperation among the e-services is necessary, so as to provide the MI library with the necessary data.

As described with reference to Fig. 1, e-services sometimes form a tree or a directed acyclic graph, with the different e-services being the nodes and conversations among e-services being the branches. As these trees are created (either statically or dynamically) and as conversations/transactions are performed, management information can be exchanged among the e-services 50, 62 and 64. A protocol in accordance with the invention may be referred to as the "MI protocol," because management information is exchanged among the e-services. Fig. 4 illustrates one embodiment of a header (i.e., tag) that is appended to an outgoing document of an e-service. The header is a management information (MI) structure 72 that includes a correlator 74 and a management information object (MIO) 76. The correlator is comprised of correlation information that uniquely identifies a transaction. As will be described more fully below, in one embodiment the correlator includes a transaction class identifier, a transaction handle that is an incremented integer, and an interaction handle that is an incremented integer that corresponds to an interaction with a sub e-service. The transaction class identifier and the handles combine to identify a particular conversation. The interaction handle of a specific transaction instance is used to identify the sub e-service's contribution to the transaction's statistics.

The MIO 76 of the MI structure 72 consists of a policy 78 and management parameters 80. The policy information may initially be provided by a client. This parameter represents the expectations of a consumer or an e-service provider regarding how a service request should be handled. As one example, a request may be prioritized as either "normal," "silver," "gold," or "platinum." The policy information may include a time constraint and/or other constraints relating to the processing of the requests. The management parameters 80 may be considered to be operational parameters of an e-service that is providing the information. As examples, the management parameters may include an e-service health index, an e-service availability indicator, an e-service reliability indicator, an e-service performance indicator and/or fault information. Preferably, at least two of these five categories of information are included. The e-service health index indicates the current status of an e-service, with possible values being "up," "down," "congested," "halted," "restarting" and "unknown." The availability indicator represents the current or projected availability of the e-service. The availability indicator may be placed in terms of downtime per a designated period of time, with downtime being the duration of e-service unavailability as a result of system or application faults. The reliability indicator may be a measure in terms of fault rate, such as the ratio of faults to service requests. Examples of the performance indicator include response time, response status, throughput/rate, aborted count/rate and failed count/rate. The response time is the duration from the sending of a request to the receiving of a response. The response status indicates if a service request has been responded to, aborted, or failed. The throughput/aborted/failed rate is the number of committed/aborted/failed services per the number of service requests. Finally, a fault parameter contains the information about a fault when a request is serviced. Examples of the fault parameter include a fault identifier, a fault description, and the time at which the fault occurred.

The identified management parameters 80 form a base for the management of e-service conversations. Depending upon how much information one e-service is willing to expose to another, some of the raw parameters can be put into the MIO 76. Thus, the aggregated information can be calculated at the receiving e-service on the basis of the raw parameters. When the raw parameters are provided, the aggregated information need not be placed within the MIO 76. As one example, the MI library 68 of Fig. 3 can locally compute a throughput rate for a remote sub e-service by using a formula such as dividing the number of successful responses by the number

of total responses from the sub e-service. The MI structure 72 of Fig. 4 is created by default by the MI library 68 of Fig. 3. If the MI library is a component of an intermediate e-service within a succession of e-services, the management parameters 80 are updated to reflect the local management parameters, but the pre-existing management information is maintained. Similarly, the information within the correlator 74 is updated, so that the parent e-service receives a response document that is tagged with an MI structure containing the full correlator of transaction flow information.

Referring briefly to Fig. 5, the e-services at each level can further enrich the data collection by agreeing upon a certain e-service specific MIO (ESSMIO) 82 that is added to the MI structure 72. The ESSMIO is defined in a particular schema that enables the collaborating e-services to furnish business logic data, such as the number of book-buy requests that are received. In the same manner as the MIO 76, the ESSMIO is updated at each service to generate a tree which is tagged to a document to be transmitted to another e-service or to a client.

Using the MI structure 72 of Fig. 4 or Fig. 5, when an e-service conversation traverses multiple e-services, the correlation information of the correlator component 74 is required to track the component and to correlate the management information within the MIO component 76 contributed at each participating e-service. As the conversation is executed, the MI structures 72 are correlated and updated, so that they may be tagged to a response document. The management parameters at each level are collated into a single MI structure which is tagged to the final response to the parent e-service. That is, when an e-service transmits a response to its parent at the middle or the end of a conversation, the piggybacked MI structure contains the management parameters relative to the sub e-services involved in the conversation.

An e-service conversation typically begins with a request from a consumer or a parent e-service and ends with a response or a new request to another e-service. In between, one or more interactions between two e-services or between an e-service and a client may be triggered, depending upon how a conversation is defined and implemented. The possible interactions include:

The receiving e-service replies with an immediate response to the initial request. This will be an http response. The actual business-level reply is then sent asynchronously at a later time;

The receiving e-service can receive one or more subsequent requests regarding the same conversation;

The receiving e-service can send one or more requests to at least one remote sub e-service; and

5 The receiving e-service can receive one or more responses from the remote sub e-services.

In order to manage an e-service conversation with the above communication patterns, two issues must be addressed. First, the communication between e-services is "asymmetric." That is, one or more responses
10 can correspond to one request, or a single response may correspond to multiple requests. It is also possible that a responding e-service will send its response directly to the ultimate requester, rather than to its parent. Second, the request and response documents that are sent and received by an e-service can be handled by different threads/components, while each
15 thread/component handles the documents relative to different conversations. To address the two issues, some form of correlation is needed. In accordance with the invention, a set of APIs is used for generating, exchanging, and syndicating MI structures at the cooperating e-services. As a result, an e-service, when receiving a request or response, can associate the document with a specific conversation and can identify a response with a corresponding request. MI APIs enable getting and setting of the management
20 information objects 76 described with reference to Figs. 4 and 5.

Two types of information are collected from e-services. One type is descriptive information, which contains the e-service and transaction
25 definitions. The other is the management information, which contains the data for each instance of a conversation. The management information is part of the MI definition which will be described below.

The descriptive information is provided in the class
MITranRegistration. An e-service, when getting started, initiates an instance
30 in this class and calls registration method MI_register_transaction to define the mapping from a universally unique transaction class ID to a name pair (service, transaction). There are two versions of MI_register_transaction. The version that is used in a particular case depends upon whether the interaction ID is provided as a parameter or is generated by the MI library 68
35 of Fig. 3. The MI library maintains a list of registered transactions. When the method MI_register_transaction is called, a new entry is added in the list. The definition of the class is as follows:

```

public class MITranRegistration extends Object {
//Public Constructors
    public MITranRegistration();
//Public Instance Methods
    public short MI_register_transaction(
5          String service_URI,
          String tran_name,
          byte[] tran_id;
          int flags);
    public byte[] MI_register_transaction(
          String service_URI,
          String tran_name,
          int flags);
10 //misc ...
}

```

The management information is provided in the class **MIServiceTran**. This class represents e-service transactions when they execute. An e-service creates as many instances as it needs. Typically, this is at least as many as the number of transactions that can be executed by the e-service simultaneously. An e-service can create a pool of **MIServiceTran** objects, can take one such object from the pool to use when a transaction starts, and can return the object to the pool after the transaction ends, so that the object can be reused. Internally, each entry in the list of registered transactions has a list of transaction instances. Each entry contains the MI structure of a specific transaction instance. When a new instance of a transaction class is started, a new entry is allocated and added in the corresponding instance list. As previously stated, the contained MI structure is updated at each stage of the transaction. The maximum length of a transaction instance list depends on the system configuration. The definition of the class **MIServiceTran** is as follows:

```

public class MIServiceTran extends Object {
//Public Constructors
    public MIServiceTran();
30 //Public Instance Methods
    public Policy MI_getPolicy
        (Document document
         int doc_type);
    public long MI_start(byte[] tran_id,
        long ptran_handle;
        ESSMIO essmi_object,
        int flags);
35    public long MI_start(byte[] tran_id,
        long ptran_handle,
        long tran_handle,
        ESSMIO essmi_object,
        int flags);

```

-12-

```

public long MI_update(long tran_handle,
                      ESSMIO essmi_object,
                      int flags);

public Document MI_sendReq
5      (Document document,
        int doc_type,
        long tran_handle,
        Policy req_policy,
        ESSMIO essmi_object,
        int flags);

public long MI_recvReq
10     (Document document,
        int doc_type,
        long tran_handle,
        ESSMIO essmi_object,
        int flags);

public Document MI_sendRep
15     (Document document,
        int doc_type,
        long tran_handle,
        ESSMIO essmi_object,
        int flags);

public long MI_recvRep
20     (Document document,
        int doc_type,
        long tran_handle,
        ESSMIO essmi_object,
        int flags);

public long MI_stop(long tran_handle,
                    int tran_status,
                    ESSMIO essmi_object,
                    int flags);

//misc for getting MIO and ESSMIO...
}
25

```

Fig. 6 is a process flow of steps that are followed when a request is received from a parent e-service or from a consumer. At step 84, the request is received at the e-service. While not critical, the request is typically an XML document. As a result of receiving the document,

MI_getPolicy is invoked to retrieve from the document the policy containing the information regarding how the service request should be handled from the perspective of the client. Thus, at step 86, the handling policy is determined.

At step 88, the method MI_start is used to initiate a transaction. There are two versions of MI_start, with the selection of the version being determined by whether a transaction handle is provided by the user or is generated by the MI library 68 of Fig. 3. The other parameters of this method include the transaction class ID, the parent transaction handle, and the ESSMIO that was described as an optional component when referring to

Fig. 5. The transaction class ID is the type of transaction to which the particular transaction instance (the transaction handle) belongs. The parent transaction handle associates this transaction with its parent transaction, if any. The ESSMIO is used to contain the business logic information that is relevant to the transaction, if any.

If the transaction is a result of receiving the service request from a remote e-service, the method MI_recvReq is called at step 90 by providing the transaction handle and the received document. MI_recvReq retrieves the MI and ESSMIO trees from the document header (tag) at step 92 and creates a new MI structure at step 94. In Fig. 5, the new MI structure 72 will include the correlator 74 and the management information object 76 for the particular transaction instance. Then, MI_recvReq updates the MI and ESSMIO trees at step 96 by appending the transaction's MI/ESSMIO to the MI/ESSMIO trees retrieved from the document and by associating the paths with the transaction instance. The resulting MI and ESSMIO trees are used to identify this class instance in the context of the containing e-service conversation. Moreover, the trees present the corresponding management statistics. The correlator 74 for the transaction instance is made unique by the combination of (1) the transaction class ID, (2) the transaction handle which is an incremented integer, and (3) the interaction handle which is an incremented integer and which corresponds to an interaction with a remote sub e-service. The ID and handles of transactions participating in a conversation are combined to identify the conversation. The interaction handles of a specific transaction instance are used to identify the remote sub e-service's contributions to the transaction's statistics.

As previously noted, an e-service can receive a second request from a parent e-service during a transaction. In decision step 98, a determination is made as to whether a subsequent request is within the same transaction. If yes, the process returns to step 90, so that MI_recvReq is again invoked by providing the same transaction handle, the new received document, and the updated ESSMIO of the transaction, if any. In step 96, the MI tree and the ESSMIO tree are again updated with the retrieved and passed information.

Fig. 7 is a process flow of steps that are followed in sending a service request to a remote e-service that is a sub e-service. The process may be triggered by either receiving a request from a parent e-service at step 100 or by receiving a response from the remote e-service at step 102, where the response requires follow-up. If the triggering action is the receiving of the

request from the parent e-service at step 100, the receiving e-service follows the sequence of steps that were identified in Fig. 6, as represented by step 104 in Fig. 7. Referring briefly to Fig. 6, the decision step 106 determines whether a request is needed to be sent to a remote e-service, so that an affirmative response at step 106 causes the method MI_sendReq to be invoked at step 108, which is shown in both Figs. 6 and 7. On the other hand, if the process is triggered by receiving the response from the sub e-service at step 102 of Fig. 7, the method MI_rcvRep is called at step 110. The process that is invoked by calling MI_rcvRep will be described later with reference to Fig. 8. For either triggering process, the step 108 is reached.

At the step 108 of invoking MI_sendReq, the parameters include the request document to be sent, the transaction handle, and the updated ESSMIO, if the transaction's e-service specific management information has changed. A policy can also be provided to indicate how the request should be serviced. The transaction handle is used to locate the MI and ESSMIO trees of the transaction, as indicated at step 112. If a policy and/or an updated ESSMIO are supplied, MI_sendReq will use them to update the MI structure accordingly. The MI_sendReq method is invoked to perform two tasks. First, the MI library 68 of Fig. 3 records the start of a new service interaction for this transaction instance. Second, the proper MI and ESSMIO trees for the outgoing request document are generated. The two tasks are represented by the update step 114 in Fig. 7. The resulting MI structure includes the management information and ESSMIO for the transaction instance and includes all of the information from predecessor e-services. However, the MI structure excludes such information from remote e-services with which the transaction previously interacted.

The MI structure is appended to the request document at step 116. The MI structure is a tag that includes the generated MI and ESSMIO trees. The tag is separately accessible from the request document, since it is appended as header information. The request document and its tag are then transmitted at step 118.

Fig. 8 represents the process flow of steps that are followed when a response document is received at an e-service. At step 120, the response and its tag (MI structure) are received from a remote e-service via the Internet. As a result, the method MI_rcvRep is invoked at step 122. Execution of this method retrieves the MI and ESSMIO trees at step 124. These trees contain the latest conversation path and the management statistics relevant to the latest interaction. Then, the method uses the

transaction handle to locate the entry of the transaction in the local MI library (step 126). The end of the interaction is marked at step 128 and the statistics of the transaction are updated at step 130, using the received and retrieved management information and ESSMIO.

5 An e-service may send back some preliminary or final result before or at the end of a transaction. If this occurs, the current conversation path and its statistics at each participating e-service tier need to be send back with the response document. This is done within the appended tag of the response document. Also, the context of the transaction needs to be
10 updated, so that any subsequent response will not contain the statistics and e-service interactions prior to this point. To achieve this, method MI_sendRep is invoked by providing the document to be sent, the transaction handle, and the ESSMIO of the transaction, if updated. With the transaction handle, MI_sendRep locates the context of the transaction, provides the
15 update, and summarizes its statistics. Then, the method generates the MI and ESSMIO trees containing the statistics of the transaction and the break-downs at each participating e-service tier. Finally, MI_sendRep inserts the generated MI and ESSMIO trees into the appended tag of the response document before they are returned.

20 The method MI_end is used to inform the MI library of the end of a transaction. Moreover, the method summarizes the statistics. The MI library is informed of the status of the completed transaction. If the application logic finishes successfully, the status is set to MI_GOOD. On the other hand, if the service request is not satisfied (e.g., the service could not reserve
25 a hotel room due to a lack of vacancies) the status is set to MI_ABORTED. The status is set to MI_FAILED if there is an application or a system failure (e.g., a sub e-service was unavailable).

 The APIs provided by the MI library 68 of Fig. 3 can be classified into three categories: registration, demarcating business transactions,
30 and tracking interactions/conversations with the outside world. When an application/e-service starts, it calls MI_register_transaction to register the types of transactions it provides. During runtime, MI_start begins an instance of a registered transaction, while MI_stop is called to end the transaction instance. In between, the application/e-service calls MI_sendReq,
35 MI_rcvReq, MI_sendRep, or MI_rcvRep when sending or receiving a request or response.

 Fig. 9 shows a simplified example of operations in which a client 132 has a conversation with a server 134 in order to place an order.

Information regarding the order is transmitted in a request document 136 having separately accessible information within an appended MI structure 138. Upon receiving the request document, the server invokes methods MI_register_transaction and MI_recvReq, as described with reference to
5 Fig. 6. Actions are then performed on the basis of business logic. The order confirmation is sent in an XML document 140 having a tagged MI structure 142. The transmission of the confirmation document involves invoking the methods MI_sendRep and MI_stop. When the confirmation document and MI structures are received at the client 132, the steps of Fig. 8 are executed.
10 This includes invoking the methods MI_recvRep and MI_stop.

A more realistic example of the invention is represented in Figs. 10–14. This example is somewhat more complex than the one of Fig. 9, but still shows only two tiers of e-services. The first tier is a travel e-service 144, while the second tier includes three sub e-services 146, 148 and 150. Inter-
15 actions with a customer are represented by input and output lines 152 and 154. Conversations between the parent travel e-service 144 and the sub e-services are represented by the bidirectional lines 156, 158 and 160. In these conversations, documents are exchanged between the travel e-service and the airline, hotel and car rental e-services.

Figs. 11–14 illustrate how the set of MI APIs is used to instru-
20 ment the transactions between the travel e-service 144 and its three sub e-services 146, 148 and 150. Although the conversations in the example are symmetric, the MI library and its API also support an asymmetric conversation model. Moreover, a transaction can cross different threads/components. In other words, the methods of MI_start, MI_recvReq, MI_sendReq,
25 MI_sendRep, MI_recvRep, and MI_stop for one transaction can be called by different threads/components.

As a result of the implementation of the invention, each of the three conversations 156, 158 and 160 includes transmissions of a tagged
30 request document 162 and a tagged payment document 164 from the travel e-service to each sub e-service and further includes transmissions of a tagged offer document 166 and a tagged confirmation document 168 from the sub e-service to the travel e-service.

Returning to Figs. 3 and 4, each MI structure 72 contains the
35 correlator 74 that was described above. The correlator identifies the context of the transaction within a conversation. It is transparently handled by the MI library 68 of the particular e-service provider 50. In one example, its schema is:

-17-

```

    <complexType name = "CorrelatorType">
      <element name = "TranID" type = "string" />
      <element name = "TranHandle"
        type = "decimal" />
      <element name = "InteractionID"
        type = "decimal" />
5    </complexType>

```

The policy 78 of the MIO 76 sets a priority for the service request. It can also set quality of service (QoS) parameters that indicate how and when the request should be serviced. An exemplary schema is:

10

```

    <complexType name = "PolicyType">
      <element name = "Priority" type = "decimal"
        minOccurs = "0" maxOccurs = "1">
      <element name = "SLO" type = "SLOType"
        minOccurs = "0" maxOccurs = "unbounded">
15    </complexType>

    <complexType name = "SLOType">
      <element name = "Term" type = "string" />
      <element name = "Constraint" type = "string" />
      <element name = "Threshold" type = "string" />
    </complexType>

```

20

Currently, the management parameters 80 in each MIO 76 contain the identity (i.e., URL for a consumer or service provider), as well as the performance, availability, and reliability statistics. A possible schema is:

```

25    <complexType name = "MeasurementType">
      <element name = "Identity" type = "IDType" />
      <element name = "Performance"
        type = "PerfType" />
      <element name = "Availability"
        type = "AvailType" />
    </complexType>

30    <complexType name = "IDType">
      <element name = "From" type = "serviceURI" />
      <element name = "to" type = "serviceURI" />
    </complexType>

    <complexType name = "PerfType">
      <element name = "RespTime" type="decimal" />
      <element name = "StartTime" type="decimal" />
35    <element name = "StopTime" type="decimal" />
      <element name = "TranStatus"
        type = "decimal" />
    </complexType>

    <complexType name = "AvailType">

```

10008402-1

```
<element name = "ReqCount" type = "decimal" />
<element name = "ComCount" type = "decimal" />
<element name = "FailCount" type = "decimal" />
<element name = "AbortCount"
type = "decimal" />
</complexType>
```

The MIO 76 consists of the policy 78 and the management parameters 80. The schema of the MIO may be as follows:

```

10    <complexType name = "MIOType" >
      <element name = "Policy" type="PolicyType"
        minOccurs = "0" maxOccurs = "1">
        <element name = "MeasuredData"
          type = "MeasurementType"
          minOccurs = "0" maxOccurs = "1">
    </complexType>

```

The correlator 74 and MIO 76 construct the MI structure 72 that is used to identify the local context of the transaction and the statistics. The schema may be as follows:

```

20  <complexType name = "MIType" >
    <element name = "ParentCorrelator"
            type = "CorrelatorType"
            minOccurs = "0" maxOccurs = "1">
        <element name = "Correlator"
            type = "CorrelatorType" />
        <element name = "MIO" type="MIOType" />
    </complexType>

```

To identify a service request in the context of a conversation, an e-service needs to access the MI structure 72 and its predecessors. For the purpose of determining how its sub e-services perform, a parent e-service needs to get the management information of the participating e-services. MITree is defined to serve this need. The structure is piggybacked on the documents exchanged among e-services. A possible schema is as follows:

```
<element name = "MITree"
        type ="MITreeType" />
```

```
<complexType name = "MITreeType" >
  <element name = "predecessor"
    type = "MIType"
    minOccurs = "0" maxOccurs = "unbounded">
    <element name = "MI"
```

```

        type      = "MIType" />
    <element      name      = "Child"
        type      = "MIType"
        minOccurs = "0"    maxOccurs = "unbounded"
    </complexType>

```

5

By aggregating and analyzing MIOs 76, the MI library 68 can provide the necessary information that higher level management agents can employ to perform a number of management tasks. Firstly, the information may be used for end-to-end conversation management. The information enables an e-service or a consumer to track its interactions with other e-services and to track the e-service flow. Secondly, the information helps an e-service perform e-service ranking and selection. An e-service can rank its remote e-service providers based on their performance, availability, and reliability. The latest statistics of the remote e-services can also help the local e-service in the selection of alternative e-service providers. Thirdly, the information enables service optimization. By identifying performance bottlenecks and service failure points, the management agents can reconfigure and/or restart the e-service to achieve better performance and availability. It also helps in e-service evolution. An e-service provider usually provides different types of services. The information provided by the MI library 68 could be used to monitor the access patterns of the provided services and to help in decisions regarding whether or how to evolve the non-performing services to generate more revenue. Lastly, the information enables consumer tracking. The information can be used to ascertain consumer patterns, so that actions can be taken to guarantee QoS for valued consumers.

Optionally, some control may be exercisable on the dissemination of management information from one e-service tier to another e-service tier. For example, depending upon any agreements between cooperating e-services, MIOs can be suppressed or added to the relevant Management Information Structure (MIS).

The protocol which has been described operates well for web-based services using messaging protocols such as SOAP and the like. Thus, the correlator information can be passed in the header of the SOAP messages exchanged between cooperative web services. This mechanism can be used for message tracking, conversation tracking, and transaction management.